Learning Neural Network Architectures using Backpropagation

Suraj Srinivas
surajsrinivas@grads.cds.iisc.ac.in
R. Venkatesh Babu
venky@cds.iisc.ac.in

Department of Computational and Data Sciences
Indian Institute of Science
Bangalore, India

Deep neural networks with millions of parameters are at the heart of many state of the art machine learning models today. However, recent works have shown that models with much smaller number of parameters can also perform just as well. In this work, we introduce the problem of architecture-learning, i.e; learning the architecture of a neural network along with weights. We start with a large neural network, and then learn which neurons to prune. We also propose a smooth regularizer which encourages the total number of neurons after elimination to be small. The resulting objective is differentiable and simple to optimize. We experimentally validate our method on both small and large networks, and show that it can learn models with considerably smaller number of parameters without affecting prediction accuracy.

Architecture-Learning The computational complexity of a neural network can be considered to be equal to the total number of neurons in the network. As a result, one may use the following objective.

$$\hat{\theta}, \hat{\Phi} = \arg\min_{\theta,\Phi} \ell(\hat{y}(\theta,\Phi), y) + \lambda \|\Phi\| \quad (1)$$

Here, θ denotes the weights of the neural network, Φ the architecture, and ℓ denotes the loss function. The λ parameter trades-off between a good fit and a low complexity model. Here, $\|\Phi\|$ denotes the model complexity, which is simply the total number of neurons in our case. We term any algorithm which solves the above problem as an *Architecture-Learning (AL)* algorithm.

Selecting width The strategy we follow to solve the Architecture Learning problem is outlined in Figure 1. To prune neurons in a layer, we multiply auxiliary gate variables to each neuron. These are either '0' or '1'. As a result, neurons with corresponding gate variables with '0' values can be pruned away. If we learn these gates along with weights, we effectively learn the width of neural network layers.

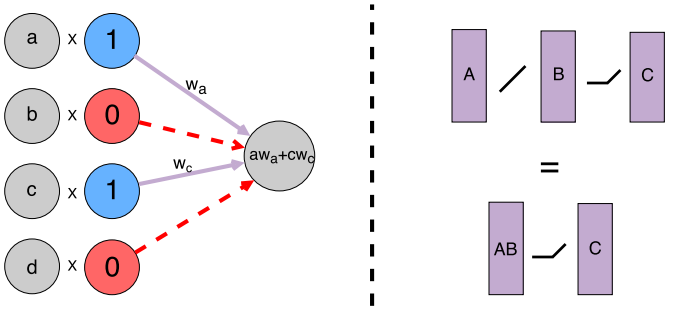

Figure 1: Our strategy for selecting width and depth. Left: Grey blobs denote neurons, coloured blobs denote the proposed auxiliary gate parameters. Right: Purple bars denote weight-matrices.

Selecting depth To reduce the depth of the network, we attempt to replace non-linearities with linear functions wherever possible. As a result, whenever linearities are present, two layers can be merged, as shown in Figure 1.

Tri-State ReLU The two disparate observations mentioned above are combined into a single non-linearity called the Tri-State ReLU (TSReLU), which is defined as follows.

$$tsReLU(x) = \begin{cases} wx, & x \geq 0 \\ wdx, & otherwise \end{cases} \quad (2)$$

The various modes of behaviour of this function for different sets of values of w and d is given in Table 1. As indicated, the proposed function can behave either can ReLU, a zero function or an identity function.

w	d	Behaviour
1	0	ReLU
1	trainable	Parameteric-ReLU [1]
0	any value	Returns 0 always
1	1	Identity function
trainable (0 or 1)	trainable (0 or 1)	Tri-State ReLU

Table 1: Various modes of behviour for different values of w, d.

Learning Binary Parameters To enable learning binary parameters instead of real valued ones, we use a thresholding operator on the gates. As a result, gate values are effectively always '0' or '1'.

$$w'_{ij} = \begin{cases} 1, & w_{ij} \geq 0.5 \\ 0, & otherwise \end{cases}$$

To enable learning binary variables toward the end of the optimization, we use the following regularizer given by $y = x \times (1-x)$. This regularizer encourages values to be close to '0' or '1', in constrast to the ℓ_2 regularizer, which encourages near-zero values.

Results We evaluate our method on both MNIST and AlexNet. For analytical experiments on MNIST, please refer to the paper. For CaffeNet, the proposed method performs compression close to state-of-the-art methods, as shown in Table 2. In addition, it also selects low complexity architectures, as shown in Table 3.

Method	Params	Accuracy (%)
Reference Model (CaffeNet)	60.9M	57.41
Neuron Pruning	39.6M	55.60
SVD-quarter-F [2]	25.6M	56.19
Adaptive FastFood 16 [2]	18.7M	57.10
AL-conv-fc	19.6M	55.90
AL-fc	19.8M	54.30
AL-conv	47.8M	55.87

Table 2: Compression performance on CaffeNet.

Method	Architecture
Baseline	96 256 384 384 256 4096 4096 1000
AL-fc	96 256 384 384 256 1536 1317 1000
AL-conv	80 127 264 274 183 4096 4096 1000
AL-conv-fc	96 256 384 384 237 1761 1661 1000

Table 3: Architectures learnt by our method.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[2] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. *arXiv preprint arXiv:1412.7149*, 2014.